BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2019 Bc. Jonáš Bräuer

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS
ÚSTAV TELEKOMUNIKACÍ

CLOUD COMPUTING
VÝPOČTY V CLOUDU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Jonáš Bräuer**
AUTOR PRÁCE

SUPERVISOR — **M.Sc. Sara Ricci, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jonáš Bräuer ***ID:*** 162522
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Výpočty v Cloudu

POKYNY PRO VYPRACOVÁNÍ:

Student nejdříve nastuduje současný stav problematiky cloudových výpočtů (Cloud Computing) a technik rozdělení dat (Data Splitting). Výstupem diplomové práce bude funkční implementace dvou původních protokolů [1] a [2] pracujících ve scénáři “non-colluding honest-but-curious” a jejich modifikovaných verzích pro scénář „colluding honest-but curious“. Všechny protokoly budou implementovány v jazyce Java (nebo C++), vzájemně srovnány a vyhodnoceny z pohledu výpočetní složitosti.

DOPORUČENÁ LITERATURA:

[1] Domingo-Ferrer, J., Ricci, S., Domingo-Enrich, C.:

Outsourcing scalar products and matrix products on privacy-protected unencrypted data stored in untrusted clouds. Information Sciences, vol. 436-437, pp. 320-342, (April 2018).

[2] Nassar, M., Erradi, A., Sabry, F., Malluhi, Q. M.: Secure outsourcing of matrix operations as a service. In IEEE CLOUD 2013, pp. 918-925. IEEE (2014).

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: M.Sc. Sara Ricci, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRACT

Security and privacy preservation of data stored on public cloud platforms is a widely addressed topic nowadays. In this thesis we explore the possible use of data splitting technique as opposed to more traditional encryption of sensitive data. Goal of this thesis is to implement and compare two already proposed protocols for matrix product computation, that use two different data splitting approaches. In addition, from these two protocols, working in non-colluding scenario, an original variant was proposed, adding colluding scenario compliance. All three protocols have been implemented using Java platform and real public cloud providers, and both performance and storage requirements have been measured

KEYWORDS

cloud computing, data splitting, data security, web server, Java

ABSTRAKT

Bezpečnost a zachování důvěrnosti dat uložených s využitím veřejných cloudových služeb je dnes velmi aktuální téma. V této práci se, na rozdíl od tradičního šifrování, zabýváme možností využití dělení dat k zabezpečení citlivých dat. Cílem práce je implementovat a porovnat dva již publikované protokoly pro násobení matic využívající dva rozdílné přístupy dělení dat. Na základě jejich vlastností byla navržena originální varianta, která na rozdíl od původních protokolů zohledňuje i případ, kdy poskytovatelé cloudových služeb tajně spolupracují. Všechny tyto protokoly byly naimplementovány za použití platformy Java a veřejných cloudových služeb. Na závěr byla změřena výkonnostní náročnost a zhodnoceny požadavky na úložiště.

KLÍČOVÁ SLOVA

cloudové výpočty, dělení dat, zabezpečení dat, webový server, Java

BRÄUER, Jonáš. *Cloud computing*. Brno, 2019, 56 p. Master's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by M.Sc. Sara Ricci, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Tato práce se zabývá problematikou zabezpečení dat uložených na veřejných cloudech. Dokud byla využívána privátní infrastruktura, zajistit bezpečnost citlivých dat nebylo složité. S příchodem optimalizovaných, a tudíž značně výhodných cloudových platforem se ale situace mění. Motivace využít tyto služby jak k úschově dat, tak k výpočtu výkonově náročných operací je vysoká. Garantovat bezpečí a důvěrnost takto zpracovaných dat je však poměrně obtížné. Vzhledem k veřejné podstatě těchto služeb je nezbytné znemožnit čitelnost dat, klasické postupy šifrování zde však nejsou tak výhodné. Data je totiž nutné opětovně stahovat a dešifrovat předtím, než je lze použít. Právě tuto významnou výkonnostní nevýhodu se snaží kompenzovat alternativní přístup – tzv. dělení dat. Bezpečnost a důvěrnost je zde dosažena fyzickým a geografickým rozdělením částí původního datového souboru. Dílčí data jsou tedy uložena na veřejné cloudové platformě zcela čitelně, zle zároveň tak, aby nebylo možné nikterak vyvodit citlivé vazby původního souboru. Výhodou jsou téměř nulové nároky na režii při přístupu k datům a zejména možnost provádět výpočty (např. statistická analýza) bez nutnosti stahování dat nebo jejich dešifrování.

Dělení dat je ve své podstatě velmi efektivní metoda, je však nutné dodržet učité bezpečnostní podmínky. Vzhledem k tomu, že data jsou plně svěřena do rukou poskytovatele cloudových služeb, je nezbytná obezřetná volba této služby. I za předpokladu, že se jedná o významného poskytovatele s vysokým stupněm zabezpečení infrastruktury, měli bychom při návrhu volit více skeptický přístup. Výchozí scénář tzv. *čestného, ale zvídavého* poskytovatele anticipuje, že poskytovatel se může pokoušet data analyzovat. Protokoly zajišťující důvěrnost dat tedy musí dělení provést tak, aby jednotlivé fragmenty samotné neobsahovaly žádné citlivé informace. Přestože tento scénář je pro většinu případů postačující, v této práci navrhujeme protokol poskytující důvěrnost i v případě scénáře mnohem náročnějšího. V případě, že jednotliví poskytovatelé znají lokace ostatních fragmentů a jsou schopni si k nim (např. kolaborací) zajistit přístup, metoda dělení dat ztrácí svůj význam. Pro tento případ je nezbytné nějakým způsobem skrýt, které fragmenty spolu tvoří původní datový soubor. Tato informace musí zůstat velmi bezpečně uschována, podobně jako privátní klíče asymetrických kryptosystémů.

V rámci praktické části této práce byla provedena implementace služby simulující uložení vstupního datového souboru za pomocí dvou specifických metod dělení dat. Technicky se jedná o privátní proxy server, překládající dotazy uživatelů privátní sítě na dotazy směrem ke cloudovému rozhraní, kde jsou již data rozdělena. Vzhledem ke specifičnosti těchto překladů je nutné vytvořit od základů i odpovídající cloudové rozhraní. Toto rozhraní je zveřejněno za pomocí cloudových platforem. Více takto nastavených koncových bodů spolu s proxy serverem tvoří komunikační síť, která bude vykonávat distribuované aritmetické operace.

Dalším krokem tedy je, dle již publikovaných prací, implementovat dva protokoly pro výpočet násobku dvou matic, tedy simulace výpočtu korelační matice vstupních dat. S ohledem na to, že tato operace vyžaduje celý soubor, tedy všechny fragmenty, byla pro názornost srovnání implementována i operace simulující přístup pouze k jednomu fragmentu. Zde se jedná o potenciálně užitečnou informaci aritmetického průměru jednoho z atributů původního datového souboru. Na základě těchto protokolů byl navržen třetí originální protokol, poskytující přidanou bezpečnost i v případě, kdy jednotliví poskytovatelé cloudových služeb spolupracují. Výsledná služba byla vytvořena za pomocí ekosystému Java EE, za použití technologií JAX-RS, EJML, Apache Tomcat v kombinaci s cloudovými platformami Amazon Web Services, Microsoft Azure a Google Cloud.

Pro zhodnocení a porovnání výkonnosti jednotlivých protokolů bylo provedeno měření za pomoci předem zpracovaných reálných i náhodných dat. V případě reálných dat se jedná o sbírku 163 066 amerických nemocničních zařízení a některých jejich potenciálně důvěrných, finančních a statistických údajů. Pro lepší zhodnocení výkonnostní závislosti na množství dat bylo měření provedeno také s náhodně vygenerovanými daty. Postupně byly použity datové soubory s 10 sloupci a 10 až 300 000 řádky. Kromě výkonnostní náročnosti byly zhodnoceny také celkové požadavky na úložiště a operační paměť.

Z výsledků měření můžeme jednoznačně vyvodit, že protokol využívající metodu aditivního dělení dat je v případě výpočtu korelační matice nejrychlejší. Druhý původní protokol, využívající efektivnější vertikální dělení, je poněkud složitější, a proto výkonnostně zaostává. Jeho velikými výhodami jsou ovšem větší bezpečnost vůči odposlechu, významně nižší nároky na úložný prostor a především možnost provádění výpočtů na fragmentu samotném (například aritmetický průměr sloupce).

Na základě tohoto protokolu tudíž byla navržena varianta s přidanou permutací fragmentů před odesláním na cloud. Tato permutace zpřetrhá vazby mezi řádky jednotlivých fragmentů. Bez znalosti této permutace tedy nejsou dotyční poskytovatelé schopni původní datový soubor zrekonstruovat ani v případě jejich spolupráce. Spolu s touto přidanou permutací bylo nutné značnou část funkcionality, kterou v případě původního protokolu vykonává cloudový server, provést lokálně. To ovšem způsobilo významné snížení efektivity protokolu. Výhodami však zůstávají vlastnosti původního protokolu a zejména velmi vysoký stupeň zabezpečení dat. Potenciální reálná implementace nicméně musí počítat s vyšším zatížením privátní infrastruktury.

DECLARATION

I declare that I have written the Master's Thesis titled "Cloud computing" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to offer my sincere gratitude to my supervisor M.Sc. Sara Ricci, Ph.D. for her continuous guidance, support and motivation to carry out the work. Her deep insight in the subject helped me fulfill my tasks.

Brno . . . .

author's signature

Contents

List of Figures

List of Tables

Introduction

The usage of Internet has grown exponentially in the last decade. Only from year 2012 to 2017 the total number of Internet users has more then doubled to a total of 4.2 billion which is beyond half of the world population [1]. The amount of data being transferred each month has reached (as of 2018) approximately 150,000 petabytes and is expected to almost double by the year 2021 [2], which is stunning 19.7/35.3 gigabytes per month per person worldwide respectively[3].

The data is not just being collected and transferred all over the world, but is also analyzed and processed. Usage of "big data" is literally omnipresent, as it is used in almost every aspect of our society. Analysis (e.g. text/audio/video or social media analytics) of data collected by business intelligence applications, smart meters or even wearable technology enables us to predict stock market, customer preferences or help diagnose diseases long before apparent symptoms occur [4]. When such operations need to be done on huge amounts of data sets, it generally requires a lot of computing power as well as a lot of data storage capacity. For a long time, this was done on premises and companies invested a lot of effort in establishing and maintaining complex network and server infrastructures. However, since last decade this approach is being vastly abandoned in favor of public cloud technologies.

In 2006 Amazon introduced Amazon Web Services [5] with its Elastic Compute and S3 Bucket storage applications. This was basically the first real public cloud service as users could allocate compute and storage resources in a scalable way. Other big companies followed shortly. Examples are Google with Google docs in the same year, a server farm for research purposes founded by Google/IBM in cooperation with several universities in 2007, in 2008 NASA's OpenNebula, in 2010 Microsoft with Azure, in 2011 IBM's SmartCloud and Apple's ICloud, and in 2012 Oracle Cloud [5].

By concentrating all the essential resources: computing power, storage and networking in one data center, an advanced orchestration and smart, economic usage of these resources is possible. If virtualization is employed, physical resources can be distributed and assigned even more efficiently. This phenomenon enables cloud service providers to offer their services at cost much lower than privately owned infrastructure could have ever competed with. In addition, no staff or hardware assets are to be managed, maintained, upgraded or even up-scaled as the need would arise. Adoption of cloud technologies has seen a gigantic boom among businesses worldwide. About 95% of them is using some sort of cloud strategy (with more than 20% running their infrastructure *solely* on public cloud) [6]. At first glimpse it would seem that cloud services offer but advantages to their customers, there are nonetheless quite a few drawbacks that need to be considered.

By definition, public cloud is owned by a third-party company, so no one can ever be sure in what way one's data is being stored, processed or even forwarded somewhere else. In short, there is never 100% guarantee that the data is not being misused in some way, because it is simply not owned by its owner.

Another thing that needs to be considered is a chance of an outage. Since cloud service providers aggregate a large number of customers in generally far less physical locations, the probability of such accidents is much higher. Almost all of the large CSPs (Cloud Service Provider), e.g. Amazon, Microsoft Azure or Rackspace, recently suffered one or more outages, leaving their customers offline for up to several hours with potential loss of their data [7].

At last, the most obvious and the most important threat is the possibility of external cyber attacks. Again, cloud services are – by definition – a concentration of large quantities of servers in one physical location. Moreover, the largest data centers are owned by the most influential providers thus making it very clear to a potential attacker where exactly he can strike to inflict the most damage.

Some threats are bound to happen even on private infrastructures and were happening long before public cloud computing became popular, such as data breaches or DoS (Denial of Service) attacks [8]. On the other hand, usage of public clouds introduces other vulnerabilities related solely to the public clouds themselves. Few important mentions would be [9]:

- *Cloud APIs*, being portals to the cloud hosted applications, they can contain software bugs and imperfections that can be more easily exploited now that they are publicly accessible.
- *Incomplete data deletion*, as customers don't have direct access to the resources and so data residues could be potentially left on CSP's hardware.
- *Insider abuse*, as CSP's own employees/administrators could be potential malicious actors.
- *Multi-tenancy exploits*, CSPs provide public service and thus an attacker can easily gain access into the infrastructure the same way as a customer would. There he can use advanced low-level software or hardware vulnerabilities (such as the famous Spectre and Meltdown). An example would be a cross-VM (Virtual Machine) side-channel attack [10]. Using legitimate account, an attacker is able to find location of the target and spawns a compute server instance on the same physical host. In this way, he can eavesdrop to target's communication and extract information from the shared resources such as data cache, network access or DRAM buses.

As seen above, usage of public clouds introduces new threats and thus storing sensitive data is not as safe as one might think. Of course, privacy was always an issue and common security practices involve encrypting data even if stored locally. Application of "classic" encryption to cloud environments is fine as long as the data is sitting still and is not accessed very often. But in case the data is frequently needed to be accessed and analyzed, encryption forces the user to download, decipher, make the desired operation, encrypt and upload again each and every time. This routine creates an extreme overhead and so it is not an ideal solution for public cloud scenarios. There are two different ways of avoiding this obstacle.

First, specific encryption schemes can be applied is such a way, that some mathematical operations can still be done with the encrypted data set, i.e. homomorphic encryption. Even though it eliminates the need to download/upload the data, this approach has some limitations. In case of partial homomorphic encryption, only certain operations are allowed, depending on the used cryptosystem. More importantly, the performance drawback is significant, rendering any real world application very much impossible for now [11].

Another important aspect is the ability to search for a certain piece of data. In a traditional on-premises system, database queries are very simple and effective because they mostly operate with plain-text data. However data stored in public clouds, being encrypted, seriously complicate such operations. Specially modified encryption schemes were proposed, enabling searchable encryption. These encryption schemes are however very specific and resource heavy. In addition, performing such search operations over and over may give away additional information about the stored data, thus helping in deciphering it.

The second approach is called *data splitting*. Before uploading, the data set is conveniently split into different parts in a way, that the individual parts are no longer sensitive on their own. Then, each fragment is stored at a different location. Only the data owner knows where each fragment resides and using specific protocols he can request an operation to be performed on each of the fragments. With his knowledge he can then easily reassemble the partial results. This method guarantees almost total anonymity of the stored data, enables plain-text operations and at the same time leverages the compute power of multiple cloud-infrastructure hosts.

Being not so complex as homomorphic encryption, data splitting method is more suitable for real world applications. The overhead is acceptable, no extensive data transfer is needed and the performance is drastically better and much more linear than in case of homomorphic encryption. See table 1, for an illustrative performance comparison [11].

Matrix size	10x10	100x100	200x200
Data splitting protocol	1 ms	29 ms	73 ms
Homorphic encryption protocol	2 s	5.6 minutes	40 minutes

Tab. 1: Illustrative performance comparison of data splitting and homomorphic encryption

Our Contribution

Given the presented topic, main goal of this thesis is to experimentally verify the use of data splitting to securely store sensitive data using public cloud platforms and to perform mathematical operations on them (e.g. statistical analysis). Goal is not to prove that data splitting is superior to data encryption, but instead to compare different approaches to data splitting, namely from the security point of view. As already mentioned, public clouds offer some tremendous advantages over privately owned infrastructure, but to confide a set of very sensitive data to a theoretically unknown third party entity is potentially very dangerous and needs to be compensated by additional layers of protection.

Therefore, two already proposed protocols for secure matrix multiplication will be explored in depth and implemented using contemporary technologies. With the additional use of public cloud platforms, performance drawbacks of additional security measures will be evaluated.

Lastly, even though the selected protocols differ in security approaches, both of them are not able to comply with the most demanding security scenario of colluding cloud service providers. That is why we will propose an alteration, that is an enhanced protocol, compliant with colluding scenario. This protocol will be implemented in the same way and compared to the original ones.

1 Background

In this chapter, we will introduce the studied problematic and depict the scenario. In section 1.1, theoretical background of cloud computing will be briefly explained. Then, in the following sections, proposed architecture and its components will be described along with the security model of cloud service providers.

1.1 Cloud Architecture

Even though cloud computing is an enormous success and transformed the whole paradigm of IT (Information Technology) business operations, it basically didn't bring anything new to the table. The technologies involved in the big CSP's infrastructures are still the same as in case of privately owned infrastructures. The networking protocols, APIs (Application Programming Interface) or database backends still work the same, the fundamental change is however in how we access these public cloud services [13].

In essence, there are three approaches of how a CSP can provide his cloud services. First, being the most traditional way, is the provisioning of raw hardware units - Infrastructure as a Service (IaaS). These units, such as compute/storage servers or networking elements such as firewalls can be dynamically allocated or released and are mostly virtualized to achieve greater performance. In this scenario, customer has to manage, on his own, every aspect of the logical infrastructure. Still a significant benefit since there is no need for physical infrastructure management as well as initial investment.

The next logical step is to predefine and preconfigure the most used setups and offer them as a service - PaaS (Platform as a Service). In this way, enterprises gain access to specific functions and frameworks with the beloved "elastic" benefit of cloud services. Developers can use these platforms to quickly deliver and test applications and even deploy them without worrying much about the physical server machines, their configuration or maintenance. This approach is the synonym for the agile methods of development and leverages the benefits of technologies such as virtualization, containers (Docker) and orchestration (Kubernetes).

Last final step is to provide not just framework, but the whole application. SaaS (Software as a Service) means that CSP manages both physical and logical infrastructure himself and offers only the finished, deployed software. Depending on the target audience, these services can vary from cloud storage and document sharing, through whole enterprise resource planning ecosystems up to telecommunication or tele-presence applications. The use and development of such solutions has seen a huge growth and all major CSP's have and promote them heavily, e.g. Google

Apps, Microsoft Office 365, Salesforce, DropBox and many more. Also "traditional" software companies had to keep up the pace and adapt their business models. The transition phenomenon of the "software + deployment + support" model to pure-cloud solutions, hand in hand with subscription (pay-as-you-go) models, was literally omnipresent over the past few years.

In this thesis, we are particularly interested in IaaS. In our scenario, data stored at a number of different CSPs will be accessed via a specific API. This API is not advertised or pre-made by the provider. Hence, in our case, our tailored server application will be manually deployed on a virtual compute server instance, provided by the CSP.

1.2 CLARUS

In our scenario, the goal is to outsource to the cloud as many heavy computations and storage capacity as possible. However, if we are to talk about secure outsourcing, the power given to a service provider has to be limited. That is why all the important program logic has to be done by a trusted entity. An idea of a trusted proxy comes to mind. This proxy service would take care of the data splitting, encrypted communication or outsourcing of the algebraic operations to the public cloud APIs.

Since there has already been a substantial amount of work done in this area, the standardization tendencies are evident. One of these emerging communities is *CLARUS* [14]. This open-source movement aims to promote the idea of sensitive data storage on public clouds. They try to achieve this with a set of standards such as various data and management interfaces for public cloud services or data storage security requirements. With these specifications in mind, Clarus created an open source project for a proxy-based solution - Clarus proxy. With such, highly modular, proxy server implementation, Clarus defines and enables a new type of service, SaaS (Security as a Service). End user is thus no longer required to take responsibility for ensuring the data is safely and securely stored.

Interesting use-case scenario, being already officially showcased by Clarus, is the application in health care industry. The problem presented is the impact of digitization era on health care industry. Data which needs to be stored is of the most sensitive kind imaginable. In addition, the amount of data to be stored is very high and thus usage of cloud services is very suitable in this particular use-case. Only problem is the unsafe, untrustworthy nature of public cloud environments. Clarus is trying to overcome this obstacle with its SaaS solution [14]. In our implementation, we will adhere to this proxy-based architecture and even though we will not directly use the Clarus proxy implementation, our solution could potentially be considered as a module for this system.

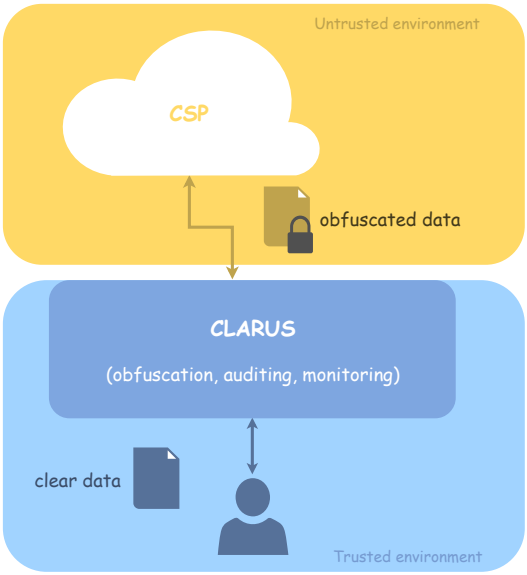

Fig. 1.1: CLARUS architecture

1.3 Security Model

Another important aspect of data outsourcing is the security model. CSPs are by default not trusted. In case of data splitting, an individual CSP is given only a fraction of the original data set, if we however consider the data to be extremely sensitive, more caution has to be made. We have to anticipate, for the sake of security, that CSPs might try to reassemble the outsourced data. From this point of view, we can differentiate between multiple scenarios [15].

Honest-but-Curious

In the first place, CSP is a service provider. He is bound by a contract to provide his services according to a previous agreement. He is solely responsible not to disregard these SLAs (Service level agreement). The fact that a CSP is abiding by these rules and doesn't try to act in any malicious way makes him "honest". We won't be considering any malicious CSPs, that might try to return incorrect computation results. Other thing however is a CSP that might be manipulating the data for his internal benefit. He might be e.g. performing analysis on the data and trying to find any interesting information (e.g. keys, passwords or literally any other personal data). In short, honest-but-curious CSP is behaving according to proposed protocols, does not try to hamper the computations or give incorrect results, but may try to analyze the data and extract as many information as possible.

Colluding and non-Colluding

From another angle, we may consider not only the initiative of individual CSPs to extract information from outsourced data, but also a cooperation between different clouds. Once a CSP recognizes, that other provider may be storing fragments of the same original data set, he may convince the other provider to cooperate and try to reassemble it. So basically we can differentiate between two types of providers [15].

- *Non-colluding CSPs*
  Multiple CSPs don't try to collaborate in any way, i.e. to pool the different data fragment together in attempt to reconstruct the data. CSPs can however analyze their individual data fragments as well as the protocol data flows.
- *Colluding CSPs*
  In this scenario, multiple CSP's may try to collude and join efforts to reconstruct the original data set. Additional techniques should be implemented to ensure that the data set is – at best – impossible to reconstruct without the knowledge of the trusted entity.

In our implementation, we will be taking into account both of these scenarios. That is: honest-but-curious non-colluding and honest-but-curious colluding CSPs. In the thesis, two versions of already proposed protocols, working in the non-colluding environment, will be modified to account for colluding CSPs. Both approaches will be compared.

1.4 Data Splitting

As mentioned earlier, there is more ways to securely store data. The more traditional and obvious way is encryption. This approach is however more resource heavy and hence more difficult to implement and use in public cloud scenarios. In this thesis, we will be focusing solely on the pure non-cryptographic solution – data splitting.

Data splitting is a method of securely store sensitive data. Original data set is separated according to predefined rules into multiple fragments and each fragment is then stored at a different location. If we consider, that the different participating sites don't know about each other and hence don't try to reconstruct the original data, we can think about it as to be securely stored.

Data Splitting

We will take into account the data as a matrix-like set of entries, with n rows and m columns. From practical point of view, the data sets are in most cases a list of entries, each representing a certain object. Columns of this table are then the individual attributes of this object. It is logical that the number of rows (from hundred thousands to millions) will greatly outnumber the usual several-to-dozens attributes. In this case, the data is said to be vertically partitioned and can represent basically any sort of database. For example hospital database, containing thousands of clients and sensitive information about their case history or detailed diagnosis. With the data formatted vertically we can then imagine three ways of splitting it [12]:

- Horizontal splitting is the simplest way of splitting the data set. Each fragment consists of number of rows, each row being the whole set of attributes of a given object.
- Vertical splitting, being more interesting in our context, gives us individual parts as a subset of attributes for all the objects.
- Mixed splitting is the combination of both previous methods. Each fragment contains x number of attributes of y objects.

If we take into account our objective – preserving anonymity – the obvious choice is the vertically split data. Even if the vertically partitioned data seems to be secure on its own, the way the data is split, i.e. along which attributes, is also very important. The possibility, that someone could link two related attributes inside one fragment, has to be considered.

In addition to the data splitting techniques that extract rows or columns form the original data set, other methods potentially can be used to split the data. The goal of this method is to divide some input data into non related fragments, so it does not matter how is this achieved. Namely *additive splitting*, used by one of the studied protocols, does not extract columns from the original matrix. Instead, it relies to the mathematical principal of addition to obtain the fragments.

Throughout this thesis, we will work only with vertically formatted data sets using vertical and additive splitting techniques in the implemented protocols.

1.5 Java & Apache

Java one of the most used languages in the world. Java is however not "just" a programming language. Java represents an entire platform. It consists of the language itself, a very extensive set of libraries and the Java Virtual Machine (JVM) which can run compiled Java byte code. This platform is further divided according to the field of use. Java for small embedded systems, Java SE (Standard Edition) and the specialized Java EE (Enterprise Edition). It contains even more additional tools and libraries, mainly for web application development. It is being used even for the most robust enterprise applications. One of the main uses for this Java is web application development. Java servlet design is the base specification for a Java server and one of its implementations along with jersey-api will be used for development of our application. The application will be targeted to Java 8 run-time environments and above.

Apache Software Foundation is a software community being active in broad spectrum of projects, mainly focused on web services. Very famous and extensively used is Apache HTTP server implementation. Similarly in "Java world", Apache foundation created a Java servlet implementation – *Apache Tomcat* [16]. It is an open source software released under the Apache License version 2. It provides an easy to use and deploy, common layer for multiple, concurrent, java server applications.

2 Our Proposal

Goal of this thesis is to implement two already proposed protocols. These will be described in detail in section 2.1. In addition, an enhanced version of these protocols, safe enough to be deployed in a colluding scenario, will be proposed (section 2.2) and implemented.

First of the original protocols, proposed in [11] (from now on *Protocol I*), is using additive splitting technique to securely divide the original fragment. Once these fragments are safely stored at a number of remote servers a series of requests is used to get partial results (multiplication terms) that, when summed up, give back the result. All the heavy computation is handled by the cloud, secure proxy does only the final addition.

The other protocol, proposed by [15] (from now on *Protocol II*), uses vertical splitting technique to outsource the original data set. This technique is quite advantageous, since data that needs to be stored is halved as well as it is not concealed at all. The operations flow, needed to obtain the multiplication result, is more complex and also the total number of requests is higher rendering this protocol more resource and time demanding.

Both original protocols are considered to be very much safe in *non-colluding* environment only. In case that individual CSPs collaborate, they can easily assemble the original data, since the fragments are stored in a completely non-obfuscated way. That is why, an additional step will be proposed in the protocol flow to prevent this direct readability and thus propose a protocol safe enough even for a colluding scenario.

2.1 Originally proposed protocols

The whole outsourcing and computational scenario will be done using one data set. The idea is for the data set to be as big as possible, vertically formatted (see 1.4) and possibly representing a sensitive source of information. To find such a specific and at the same time huge data set is very challenging. In addition, performing an arithmetic operation on two data sets is done mainly to find correlations, so it is logical that two data sets in question should be somehow correlated. Due to these facts, only one data set will be used (see 3.6) in our thesis. To simulate the correlation computation the reference operation will be $A^T \cdot A$.

2.1.1 Original Protocol I

[11] This protocol is based on additional splitting technique and thus the multiplication operation can be expressed as a trivial sum of partial terms, flow of this protocol is as follows. First step is to split the incoming data. General matrix of m rows and n columns is anticipated, that is than split using additive splitting. That is dividing the matrix into two fragments in such a way, that the sum of both of these fragments will give back the original matrix:

$$A = A_1 + A_2$$

$$B = B_1 + B_2$$

Each fragment is than securely stored at an arbitrarily chosen cloud host. The proposal recommends each fragment to be stored at two different locations for redundancy. Then, to compute the multiplication of these two data sets (matrices) following relation is leveraged:

$$AB = (A_1 + A_2)(B_1 + B_2) = A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2$$

So once the multiplication of A and B is requested, the secure proxy invokes these partial multiplications requests on each cloud host containing given pairs. If none is found, missing fragment is securely transferred from another host so that each pair is found at least once. After partial multiplication results are returned, the proxy sums all of them and replies to the original requester.

The scenario assumes, that the proxy is the only secure component and thus it holds all the responsibility. Cloud endpoints only accept computational requests and return the resulting values, with no additional logic whatsoever. Additionally, all the communication is flowing only between secure proxy and each individual cloud endpoints, they don't even need to acknowledge each others presence.

Let's consider then, that our data set X will be split into 3 separate fragments. The split operation will be done using additive splitting [11]:

$$X = A + B + C$$

If fragments are obtained using this method, our reference multiplication operation can be then expressed as follows:

$$X^TX = A^TA + A^TB + B^TA + A^TC + C^TA + B^TB + B^TC + C^TB + C^TC$$

Since:

$$A^TB = (B^TA)^T$$

We can reduce the number of requests towards cloud endpoints by reusing the the equivalent terms. So practically, we need to receive only these results:

$$\{A^TA, B^TB, C^TC, A^TB, A^TC, B^TC\}$$

2.1.2 Original Protocol II

Same as in the previous case, protocol II [15] adheres to the 'secure proxy' architecture. A trusted server translates arithmetic operation requests to the cloud hosts, where data is stored. Contrary to additive splitting, used in protocol I, here *vertical* splitting is used. An arbitrary number of columns is extracted from the original data set "as is", leaving all the data non-obfuscated. In case of a very sensitive interconnected data, another algorithm has to be used to choose how the data should be split (i.e. what columns can be present in a single fragment). Other than that, this approach is very much beneficial, because the amount of storage needed is *halved* (compared to additive splitting).

Original protocol is specified in [15], i.e. section 4, protocol 1.2. Architecture is comprised of 2 storage nodes, storing vertically split fragments, and one additional node (auxiliary node), providing one more partial operation, that effectively enables storage nodes to securely permute their fragments before sharing them. Thanks to this step, protocol guarantees total secrecy against a potential malicious eavesdropper. Steps of this protocol are as follows [15]:

1. Alice and Bob each hold one fragment of the original data x and y respectively.
2. Alice and Bob receive random vector r_x from third party Charlie.
3. Alice computes $x' = x + r_x$ and randomly permutes the values in x' to obtain $x'' = P_x(x')$.
4. Alice sends x'' to Bob and $r'_x = x'' - x$ to Charlie.
5. Bob computes $y' = y + r_y$ and randomly permutes the values in y' to obtain $y'' = P_y(y')$.
6. Bob sends y'' to Alice and $r'_y = y'' - y$ to Charlie.
7. Charlie sends $p = (r'_x)^T r'_y$ (note that p is a number) to CLARUS.
8. Bob sends $t = (x'')Ty$ to CLARUS.
9. Alice sends $s_x = (r'_x)^T y''$ to CLARUS.
10. CLARUS computes:

$$t - s_x + p = [(x + r'_x)^T y = (r'_x)^T (y + r'_y) + (r'_x)^T (r'_y)] = x^T y$$

Important differences to be noted are mainly the permutation of individual fragments and the fact, that cloud hosts (Alice and Bob) are directly exchanging information. Both these supplementary steps (*3.* for Alice and *6.* for Bob), i.e. random

number addition and vector permutation serve well their purpose. They enable the protocol to be secure enough while using randomness. A malicious eavesdropper should not be able to get any information, that could have been potentially leaked due to noise addition. This protocol is however designed in the "non-colluding" security model as Alice, Bob or Charlie know about each other. In case of cooperation between these CSPs, the protocol's security falls apart.

2.2 Proposed Variant

Both of the proposed original protocols are designed to be working only in non-colluding scenario. If we want to be sure that the outsourced data is completely safe, even in case that CSPs try to collaborate (which is even more important in case of protocol II, where cloud parties know locations of the other fragments), we have to introduce an additional operation before uploading the individual fragments. That is some sort of randomness, so that when put together, fragments will not recreate the original data.

In this thesis, we propose an additional permutation operation. For this purpose, a random permutation operation $\sigma(m)$ will be performed, each for B and C. Example of such operation would be:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

This represents a permutation operation of 3 elements. Since we work with matrices and not just vectors, each element represents one entire row. So in this example, original rows *one*, *two* and *three* will become rows *three*, *one* and *two* respectively in the permuted fragment.

$$A = \begin{bmatrix} 10 & 21 \\ 5 & 25 \\ 1 & 13 \end{bmatrix} \rightarrow \sigma(A) = \begin{bmatrix} 5 & 25 \\ 1 & 13 \\ 10 & 21 \end{bmatrix}$$

Now let's consider the original data X, split using both additional and vertical splitting. If we apply two different, random permutations on fragments B and C (e.g. σ and τ respectively), original data cannot be reconstructed with them and the criteria for non-colluding environment is thus met.

$$X = A + B + C \neq A + \sigma(B) + \tau(C)$$
$$X = (A|B|C) \neq (A|\sigma(B)|\tau(C))$$

This step however changes the outsourced data and inherently the computed value. That is why after receiving the computed value using *permuted* fragments, trusted party has to compensate for this with a reverse operation. In this case we consider the following:

$$A^T B = \rho(\sigma(A^T)B)$$

where ρ is the inverse permutation of σ. So in case of our reference multiplication operation $A^T B$, one of the two fragments can be randomly permuted (after being transposed) and then uploaded. When multiplication is invoked on the the cloud hosts, returned value will be equal to $\sigma(A^T)B$, which than the trusted proxy can easily "re-permute", using the inverse permutation, i.e. τ, giving back the right result $A^T B$.

Proposed Variant

Given our proposed permutation mechanism, protocol II is ideal to be the basis of the variant. The fact, that vertical splitting technique is used and that protocol II is already using permutation to obscure the fragments, enables us to add additional permutation before uploading with no issue. The mathematical relation still applies and the partial requests don't require the original non-permuted data. However to achieve this, the proposed layout needs to be modified. In protocol II scenario, secure proxy does not handle all the secrecy, the auxiliary node distributes the seed and compensates the random permutation by providing the third partial result. This can not be done in this proposed variant, since to provide this result, the auxiliary node needs to re-permute the received data. That is why, to be able to secure this protocol in colluding scenario, we need to move auxiliary's node functionality directly to secure proxy.

It is true that the previously load free server is now required to compute the third partial result on its own, but on the other hand, outsourced fragments can be randomly permuted before uploading them in the first place. In addition *all* the randomness is now handled by the trusted party and no auxiliary server with specific functionality needs to be deployed in the cloud.

Same as in protocol II, the original data is split using vertical splitting technique to obtain (in our case) two fragments. Then, the additional permutation operation is performed on one of the fragments and both of them are uploaded to their respective storage servers, deployed in the cloud. Secure proxy stores the permutation in its meta-data store (cache) in form of a n-rows, 2-column matrix.

Once uploaded, protocol flow can be executed with a math request to the secure proxy (CLARUS). Steps of this modified protocol, with σ and ρ as the additional permutation operation and its inverse one respectively, are as follows:

1. Alice and Bob each hold one fragment of the original data $x_{permuted} = \sigma(x^T)$ and y respectively.
2. Alice and Bob receive random vector r_x (or seed) from CLARUS.
3. Alice computes $x' = x_{permuted} + r_x$ and randomly permutes the values in x' to obtain $x'' = P_x(x')$.
4. Alice sends x'' to Bob and $r'_x = x'' - x_{permuted}$ to CLARUS.
5. Bob computes $y' = y + r_y$ and randomly permutes the values in y' to obtain $y'' = P_y(y')$.
6. Bob sends y'' to Alice and $r'_y = y'' - y$ to CLARUS.
7. Bob sends $t = x''y$ to CLARUS.
8. Alice sends $s_x = r'_x y''$ to CLARUS.
9. CLARUS computes: $p = (r'_x)r'_y$
10. CLARUS computes:

$$q = \sigma(x^T)y = t - s_x + p = [(x + r'_x)^T y = (r'_x)^T(y + r'_y) + (r'_x)^T(r'_y)]$$

11. CLARUS re-permutes the result: $x^T y = \rho(q)$

3 Implementation

Practical part consists mainly of two parts: communication architecture implementation and then, implementation of the proposed protocols themselves.

To be able to implement and compare different secure outsourcing protocol variations, we must first prepare a communication infrastructure. Using multiple cloud providers, we spawn multiple VM instances. We will be using these cloud shared VM instances as hosts for our endpoint application. On each VM, a server with our custom Java application will be running, providing a Representational State Transfer (REST) API. This API will accept *store* and *compute* requests and provide numerical results for resource heavy arithmetic operations (e.g. matrix multiplication).

Another Java servlet application, which will be hosted by a personal computer in our possession, will serve as our trusted proxy entity (as described above). Its basic functionality is to accept REST requests and according to them create connections with multiple cloud endpoints. Inside this application, individual protocols will be implemented.

As a second step (section 3.7), implementation of the proposed protocols will be done. Both original protocols as well as our proposed variant will be implemented as a module for the beforehand prepared server application.

3.1 Used Technologies & Frameworks

Basically, our proposed application is a set of web servers, communicating via REST. Since a huge amount of Internet services and their underlying server infrastructures are built in this way, there is a lot of possible approaches. There is already quite an amount of web-server implementations written in practically all of the mainstream programming languages (e.g. Pyhon/Django, Javascript/NodeJS, C#/.NET or Java/Spring and many more). Given author's capabilities and experience, Java was chosen as a language of choice.

Given the previously mentioned technologies (1.5), we will use *Apache Tomcat* as HTTP server and Java *Jersey API*, that will help with application resources definition (REST endpoints) and data formatting. Last but not least, to facilitate the implementation of proposed protocols from mathematical point of view, a set of tools for matrix algebra will be used. *Efficient Java Matrix Library* (EJML) [17] offers such toolbox wrapped in a simple to use API.

Cloud Providers

Large portion of the described application is distributed amongst multiple cloud hosted servers. For this purpose we will use three of the most common cloud service providers, i.e.: *Amazon* (AWS), *Google* (Google Cloud) and *Microsoft* (Azure). Due to academic purpose of this thesis, services from these providers will be used exclusively as part of the "free-tier" (trial) offer. At each CSP, we will create a virtual compute server instance hosting a regular *Ubuntu 18.04 LTS* operating system. It is important, that all VM instances have the same amount of resources available. In our case, we will use as our standard configuration: *1vCPU, 3.75GB RAM, 10GB HDD*, often advertised by the provider as a minimal recommended configuration. At each of used CSP's platform, we will strictly limit our spawned instances to this reference configuration to better interpret the final results.

3.2 Proposed Application

Proposed application is written in Java using above technologies. Since there is an apparent distinction between the trusted-party and cloud endpoint parts of the application, two separate projects have been created: *broker-api* and *cloud-api*. Using build tool Maven, both these projects have been built and packed into a *.war* (Web Archive) file. Each of these .war files have been uploaded to target machines, running Apache Tomcat. Three cloud-api.war packages have been uploaded to individual cloud hosts, broker-api.war package has been uploaded to a desktop PC (Ubuntu 18.04 LTS) in our possesion, to simulate the proxy server.

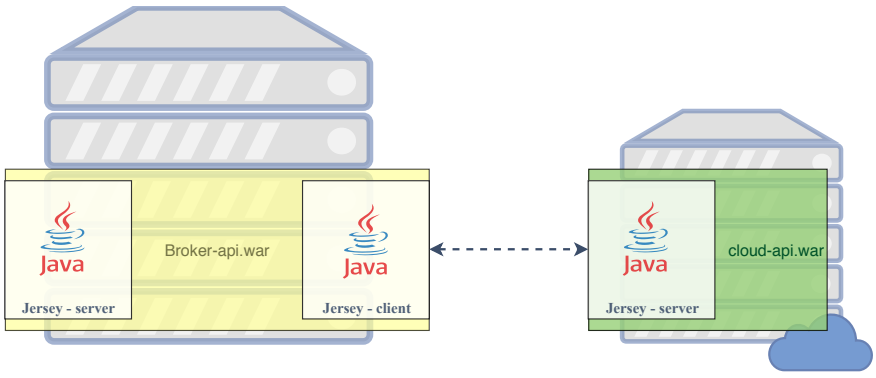

Fig. 3.1: Proposed infrastructure

3.3 Infrastructure

The scheme of our architecture topology is shown on Fig.3.2. The topology has three basic layers:

- Cloud layer, which consists of individual public cloud endpoints. Compute servers, hosted by different CSPs, will be accepting requests via their public APIs.
- In the middle resides the trusted entity. Proxy server running at the border of the secured private network, that translates requests and caches the splitting related metadata.
- At last, the secured layer, located inside the private network. Users, that will potentially want to securely outsource their data will be located here. All the external communication with our public cloud endpoints will be done via the trusted proxy server.

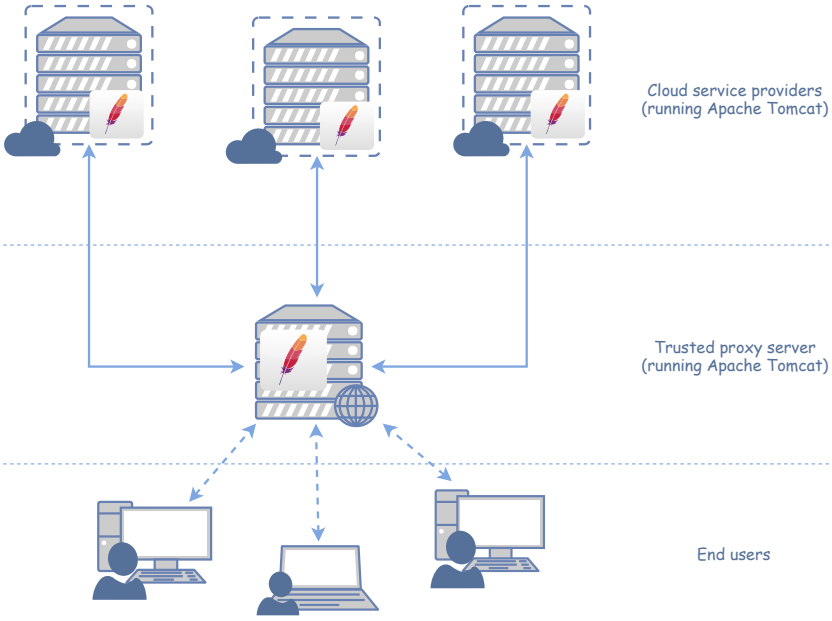

Fig. 3.2: Proposed infrastructure

3.4 REST

Due to the relative simplicity of our proposed application, we will require no more than two resources (REST endpoints), for both the proxy server and the cloud endpoints. Reflecting the two supported operations (data storage and compute operation), our communication endpoints will be as follows.

1. Proxy server will be serving these endpoints:
   - / – i.e. root, will serve HTML homepage of the application.
   - */file* endpoint will handle store requests from users (e.g. /file/upload will accept HTTP POST from HTML form).
   - */math* endpoint will accept and forward compute requests accordingly.
2. An instance of mentioned cloud endpoint will be serving these endpoints:
   - */store* endpoint will accept incoming data set fragments and store them.
   - */math* endpoint will accept and execute compute requests.

3.5 Data Formats

JSON and XML are two most used data formats used to date in REST application development. Due to its simplicity, JSON format is used throughout this project. Given the proposed endpoints, the data model was tailored to be as simple as possible. The main building block is the *matrix* data object, defined as an array of arrays of decimal numbers. This matrix object is then wrapped in the request object itself.

```
{"matrices": [
    {
      "id": "test_matrix_X",
      "data": [
        [1, 2, 3], [4, 5, 6], [7, 8, 9]
      ]
    },
    {
      "id": "test_matrix_Y",
      "data" : ...
    },
    ...
  ]
}
```

Fig. 3.3: Example of data store request JSON body

Two distinct request types are defined: *storage* and *computation*. Storage request has to contain numerical values as well as unique, human readable, matrix identifier. Computation request was designed to be as easy to use as possible, so an operation parser is implemented, that takes string representation of the operation which then gets internally translated to the series of independent cloud requests. Data formats of requests used towards the proxy server can be seen on figures 3.3 and 3.4.

```
{
  "operation": "test_matrix_X^T␣*␣test_matrix_Y",
}
```

Fig. 3.4: Example of compute request JSON body

Data format of requests towards cloud APIs are constructed in a similar way. Since human readability is not necessary, secure proxy uses Universally Unique IDentifier (UUID) for fragment referencing. In case of protocol I, cloud computation requests follow the same parsing logic as proxy computation requests. However, due to the protocol II additional complexity, cloud requests are no longer simple arithmetic operations and need to be more specific. For each specific step, data format contains the needed information, example of request from proxy to storage node is provided on figure 3.5.

```
{
  "fragmentUuid": "18da38ef-512f-47f4-8b4d-3362419cf61a",
  "otherFragmentUuid": "d8ee03f2-3331-4439-8a9b-b33686507c5d",
  "otherFragmentLocation": "http://31.156.46.101:8080/cloud-api",
  "auxNodeLocation": "http://120.187.128.35:8080/cloud-api"
}
```

Fig. 3.5: Example of protocol II cloud request JSON body

3.6 Used Data Set

As stated above, the used data set has few requirements, the main one being size. For the purpose of this thesis, one specific data set is used. It is a set of statistical data of various medical providers. It consists of several attributes, few describing the medical provider (such as ZIP code), and the rest giving the amount of discharged patients and related medical charges. The data set consists of *163 066* individual entries, with uncompressed size of 26.8 Megabytes. Data set of this size, even though not being an absolutely perfect representation of sensitive data per se, will be more than suitable for our measurement purposes. In addition, real nature of this data will allow us to obtain meaningful results.

Provider Id	Provider Zip Code	Total Discharges	Avg. Covered Charges	Avg. Total Payments	Avg. Medicare Payments
10005	35957	14	15131.85	5787.57	4976.71
10006	35631	24	37560.37	5434.95	4453.79
...	...	...	...	...	...

Tab. 3.1: Used data set example

3.7 Implementation

In this section, the already described detailed implementation of the original protocols and their variants will be explained more in depth. In theory, all the steps of the protocols were well described already, but in the actual implementation, these steps have to be tailored to the proposed architecture.

3.7.1 Original Protocol I

Protocol I is simple in its nature, so the actual implementation does not vary much from the theoretical draft. After all, additive splitting makes the protocol but a simple addition of a number of partial results. In our implementation, all the requests are executed in parallel. Since all the partial computations are equally demanding and transferred messages are of negligible size, no delays are expected and the the behaviour of the systems should be quite linear. Steps and exchanged messages can be broken into steps as follows:

1. Secure proxy reads (from its meta-data store) location of pairs: $\{A, B\}$, $\{A, C\}$, $\{B, C\}$ and sends requests for partial multiplication results A^TA, A^TB, C^TC, A^TC, B^TB and B^TC.
2. Individual cloud hosts compute the required terms.
3. All the requests are collected, missing terms are calculated according to section 2.1.1, and they are then all summed up.

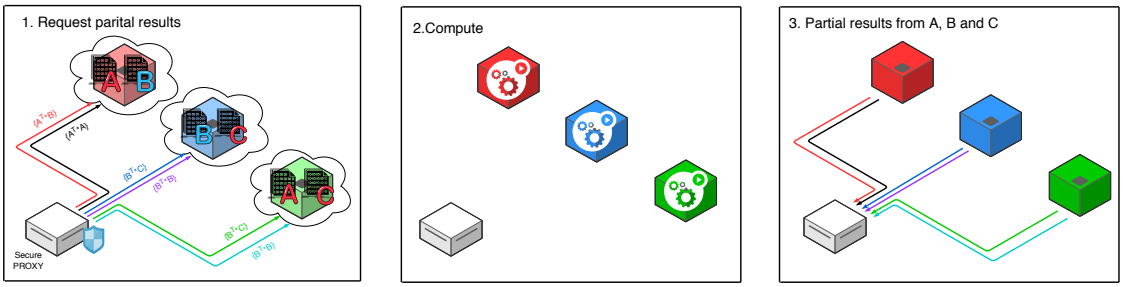

Fig. 3.6: Protocol I - stages

In our scenario, we consider the most beneficial situation for protocol I. All the fragments are uploaded in a redundant manner, so no exhaustive transferring is needed. Therefore in this case, proxy sends three requests towards three cloud endpoints, each of those makes its own two computations and replies with the result.

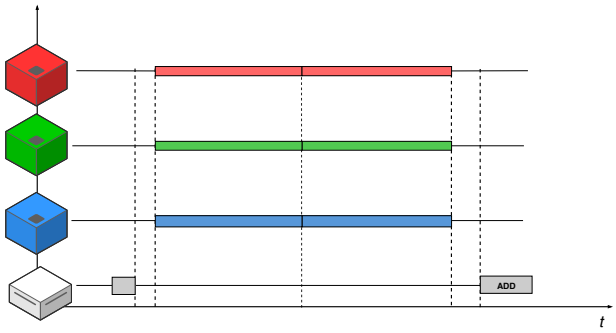

Fig. 3.7: Protocol I - time axis

Resulting matrix is squared with the same width as the original fragment (i.e. number of columns). Since our expected data set is in a vertical form, at worst tens of columns, the response body is not exceeding few kilo bytes. Overview of the protocol flow divided into individual nodes can be seen in figure 3.7. Both preparation and finalization (addition) on the proxy side is expected to be negligible given the sizes of incoming partial results. Same goes for the network related delays. At no point in time, no extensively sized messages are exchanged.

3.7.2 Original Protocol II

Protocol II is significantly more complex. The theoretical algorithm [15] is in our solution implemented as a series of triggered requests. All nodes obtain a compute request and proceed with their own requests towards the other *peer* nodes, until they are able to finalize the entire computation and reply to the proxy with the partial result. Precise steps of this protocol flow are as follows:

1. Three separate requests are sent, two for hosts containing the fragments (A, B) and the third request for the auxiliary node.
   - Request for the storage host contains this information:
     - fragmentUuid:<uuid-of-the-stored-fragment>,
     - otherFragmentUuid:<uuid-of-the-other-fragment>,
     - otherFragmentLocation:<url-of-the-other-storage-node>
     - auxLocation:<url-of-the-auxilliary-node>
   - Request for the auxilliary host contains this information:
     - fragmentAUuid:<uuid-of-the-A-fragment>,
     - locationA:<url-of-the-node-containing-A>
     - fragmentBUuid:<uuid-of-the-B-fragment>,
     - locationB:<url-of-the-node-containing-B>
2. Each of the three nodes request missing variables according to section 2.1.2, i.e. A requests y'', B requests x'' and auxiliary node requests both r'_x and r'_y from A and B respectively.
3. Two storage nodes, upon receiving the *peer* request, request a randomly generated seed (using the auxiliary node URL).

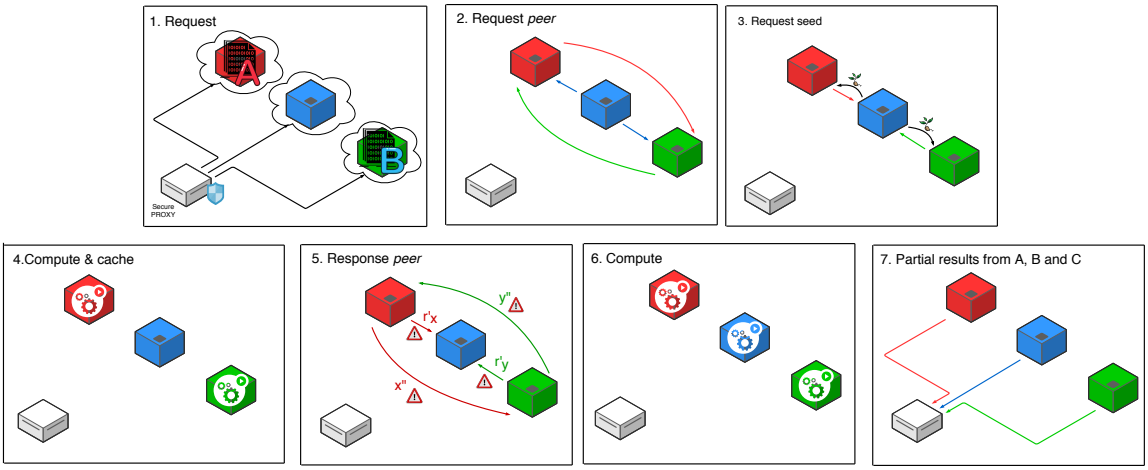

Fig. 3.8: Protocol II - stages

4. After receiving the random seed, first major computation phase is trigger at A and B. Using the obtained seed both nodes add random noise and randomly permute their fragments (i.e. obtain x'' and y''). Then,Using these, subtracted terms (i.e. $r'_y = y'' - y$ and $r'_x = x'' - x$) are computed.
5. x'' and y'' are returned to the requesting peer nodes, r'_x and r'_y to the auxiliary node.
6. Node A now has y'', node B has x'' and auxiliary node has both r'_x and r'_y. All three nodes can finish computing their partial multiplication terms and return the result to the secure proxy.
7. Proxy collects the 3 partial terms and sums them up to obtain the final result.

An important detail to notice is the exchange of intermediate results in step 5. In this step sent *peer* responses contain y'', x'', r'_x and r'_y. These are of the same size as the outsourced fragments. Considering the outsourced fragments to have more then couple of tens of mega bytes, this exchange through the public Internet could cause an important delay, depending on locations of the deployed virtual servers.

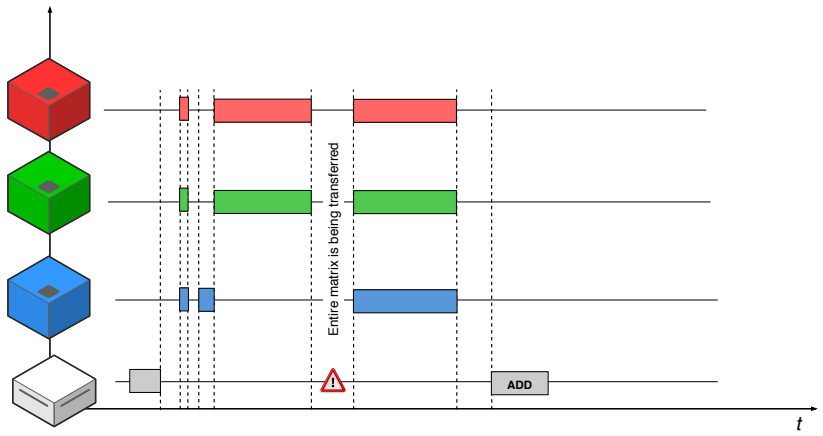

Fig. 3.9: Protocol II - time axis

Final result

Implementation of the protocol II described above enables the proxy to obtain the result of $A^T B$ operation, i.e. multiplication of the outsourced *fragments*. Request from the user however does not care about fragments multiplication, instead a multiplication of the entire outsourced data set is requested. In case of protocol I, final result – sum of the partial requests – is indeed the multiplication of the original data set, in case of protocol II however the final result for the end user needs to be composed. Given the matrix multiplication rules, in case of $A^T B$ operation (each data set split into *two* fragments), this relation can be inferred:

$$X_1 = A|B,\ X_2 = B|C$$

$$X_1^T X_2 = \left[\begin{array}{c|c} A^T C & A^T D \\ \hline B^T C & B^T D \end{array}\right]$$

In case of our reference multiplication operation, where $X_1 = X_2 = A|B$:

$$X_1^T X_2 = \left[\begin{array}{c|c} A^T A & A^T B \\ \hline B^T A & B^T B \end{array}\right]$$

Since $B^T A = (A^T B)^T$ we need only three requests: $\{A^T B, A^T A, B^T B\}$. That means in order to obtain the final result, as the end user requested, only one execution of protocol II is not enough. In addition, two more requests for $A^T A$ and $B^T B$ are needed. In our implementation, all three requests are executed in parallel, thus the load should be dispersed equally in the total elapsed time. Same principle explained here is inherently used in the protocol II variant implementation.

3.7.3 Proposed Variant II

As described in section 2.2, variation of protocol II is based on an additional permutation before data upload phase. Because of that, functionality, previously handled by the auxiliary node, now has to be transferred to the proxy server itself. This means initial request towards storage nodes already needs to contain randomly generated seed. In addition, after finalizing the partial requests, proxy needs to re-permute it using the previously stored permutation. Apart from this fact, protocol flow stays basically the same. Steps, where $\hat{\mathrm{A}}^T$ is the permuted and transposed fragment, are:

1. Two requests are sent, to each host containing the fragments $(\hat{\mathrm{A}}^T, B)$.
   Request for the storage hosts contains this information:
   - fragmentUuid:<uuid-of-the-stored-fragment>,
   - otherFragmentUuid:<uuid-of-the-other-fragment>,
   - otherFragmentLocation:<url-of-the-other-storage-node>
   - seed:<random-seed-used-to-generate-noise>
2. Two storage nodes request missing variables, same way as in the original protocol, i.e. A requests y'', B requests x''.
3. Computation is triggered at A and B. Using the obtained seed both nodes add random noise and randomly permute their fragments (i.e. obtain x'' and y''). Then,Using these, subtracted terms (i.e. $r'_y = y'' - y$ and $r'_x = x'' - x$) are computed.
4. x'' and y'' are returned to the requesting peer nodes

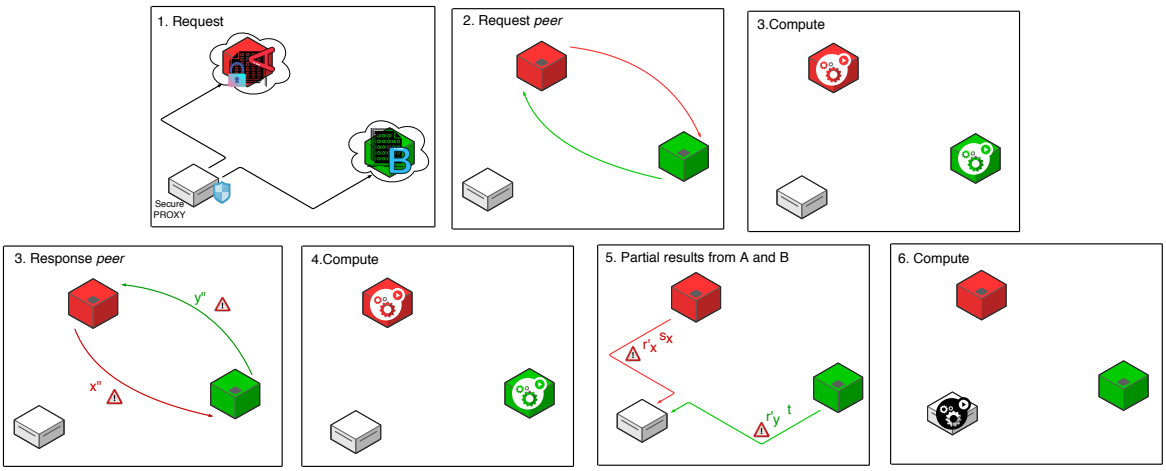

Fig. 3.10: Proposed Variant II

5. Node A now has y'' and node B has x''. They can finish computing their partial multiplication terms.
6. Both partial result and subtracted term r'_x and r'_y respectively are returned to the secure proxy.
7. Proxy collects 2 partial results and both r'_x and r'_y that it has to multiply by itself. Then the three terms can finally be summed up and the resulting matrix re-permuted to obtain the correct result.

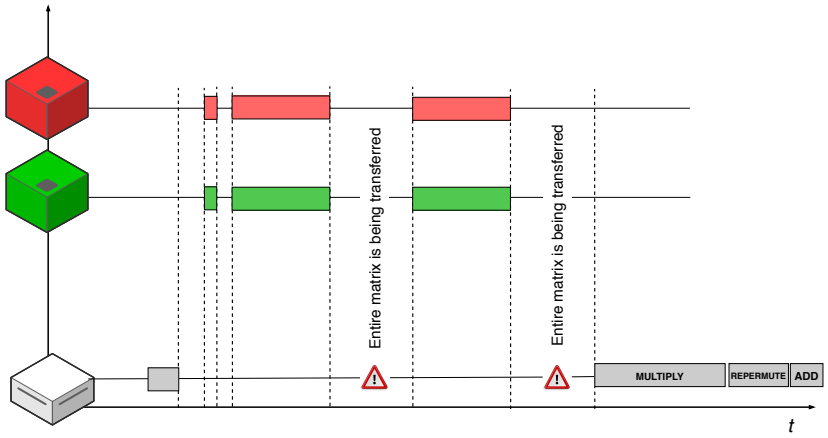

Fig. 3.11: Proposed Variant II - time axis

As we can see in figure 3.11, the total time requirements on the cloud servers do not differ much from the original protocol II. Since the computation, done by the auxiliary node, was executed in parallel, the projection into the overall elapsed time was minimal. On the other hand, the fact that we were forced to move this functionality onto the proxy server has caused this stage to be executed in series, only after the cloud nodes have finished their two computation steps.

In addition, the proxy server is expected not to be as efficient in the single thread raw computation, compared to high performance cloud servers. Also the proxy will most probably not be deployed on a dedicated server (being the case of our testing scenario), which will additionally lower the overall performance. Due to these facts, we can expect this variant of protocol II to perform significantly worse than the original protocol II itself.

4 Experimental Results

In this section performance measurements of all the implemented protocols will be discussed. Since the we are dealing with cloud outsourcing and computing, we are interested mainly in how demanding the protocol is for cloud hosts and the secure proxy server (i.e. *compute time*) and how much *disk space* is required.

Network measurements are of course of big importance, but since all the testing in our scenario is done from a private network with not guaranteed performance and with a non-representative asymmetric bandwidth, upload of even a few megabytes sized file takes a lot of time. In production environment, secure proxy server would be a very powerful machine on the edge of company's private network with virtually unlimited bandwidth in both download and upload, so the expected bottleneck of the protocol flow would still be the computation phase. In addition, splitting and storing the data is one-time operation, done once and does not influence the cloud computing protocols themselves. That is why, this part will be omitted from this final part. Network delays for message and results exchanges will, of course, be accounted for in the comparison, but will not be analyzed in more depth.

4.1 Multiplication Computation

To measure computing demands, a purely practical approach has been chosen. During the whole protocol flow, for each important computation step, time it took is measured and marked into the response body. Thus, in the final response from the secure proxy (JSON format), all these marks can be read for each node and each step of the protocol.

End-to-End

Even though all the protocols were implemented to be as optimized as circumstances allowed, it is important to note, that all the presented measurements, in their absolute values, should not be taken as final and decisive. The proposed protocols and mainly their underlying infrastructure are simulation of a production grade software, built by a team of experienced developers over a course of possibly several years. In this thesis we therefore try to accomplish rather a comparative study, of proposed protocols and the relative performance impact of additional security measures. All the protocols were implemented in the same way, using the same underlying mathematical libraries, all operations are executed as parallel as protocol flow allows. So it is very much safe to compare the protocols performances, relatively to each other.

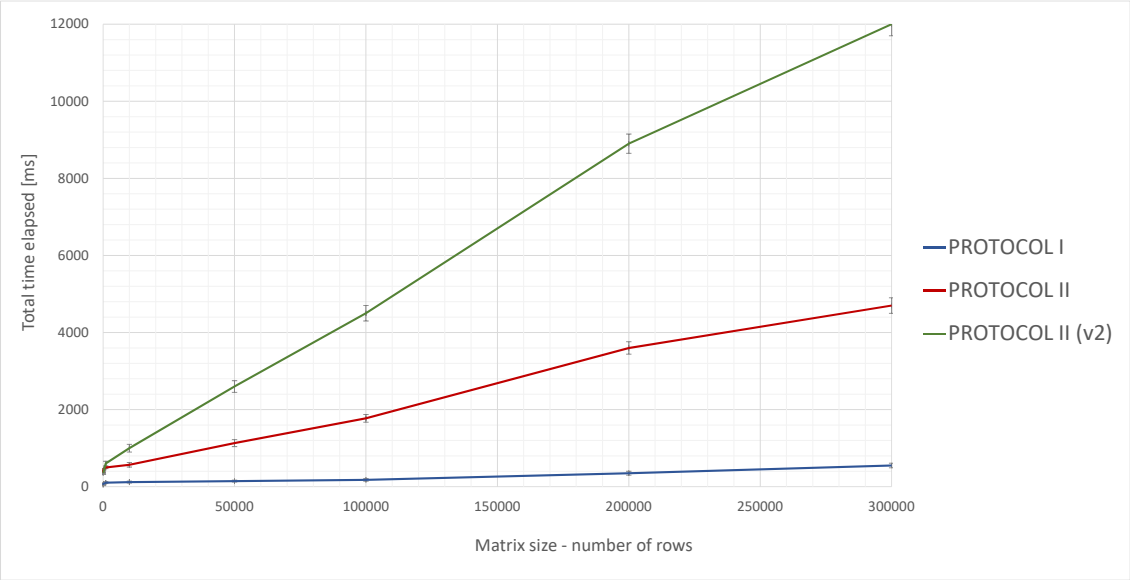

Fig. 4.1: Total time elapsed

With the help of a basic web page, served by the proxy, protocols were measured in the same way as a hypothetical client of the Security as a Service would. To better see the the linearity of proposed protocols, random matrices of fixed width 10 columns and variable number of rows were uploaded and additional measurements were taken using them. Each computation request was then sent 10 times. These values (see 4.1) were used to get an average *end-to-end* response time. Although this is not a overly precise, nor scientific method of measurement, obtained results give us an opportunity to make a clear comparison.

matrix size $[m \times 10]$	10	100	10^3	10^4	10^5	$2 \cdot 10^5$	$3 \cdot 10^5$
I	84 ± 25	87 ± 25	95 ± 25	98 ± 25	131 ± 50	210 ± 60	400 ± 60
II	370 ± 30	440 ± 30	500 ± 30	570 ± 55	1775 ± 100	3600 ± 160	4700 ± 200
II (v2)	360 ± 35	370 ± 50	600 ± 60	1000 ± 100	4500 ± 200	8900 ± 250	12000 ± 300

Tab. 4.1: Measured average elapsed time [ms] - multiplication

Brake down

End-to-end duration is the single most important thing, while taking a real life applicability into account. However, knowing more about duration of individual stages helps to give a more clear idea about the entire protocol flow. Goal of all these protocols is to outsource as much load as possible, that is why, in our implementation, duration marks for all the individual stages are part of the final response body. From these values, duration of the three most important stages was evaluated, those being:

- *Cloud* - the time portion of the process spent on the cloud side. The bigger is percentage of this step, the better.
- *Proxy* - time the proxy server spends on finalization of the compute request. This value should be as small as possible. In production environment, where many requests can be served in one instant, even a small difference counts.
- *Network* - for sake of simplicity, this value is obtained by subtracting cloud and proxy compute times from the total. Since the "total" traces are implemented at each host, we know duration from when request is accepted by the host, until response is sent out. The remaining time (Δt) is therefore quite precise and represents nicely the actual time lost due to message exchange.

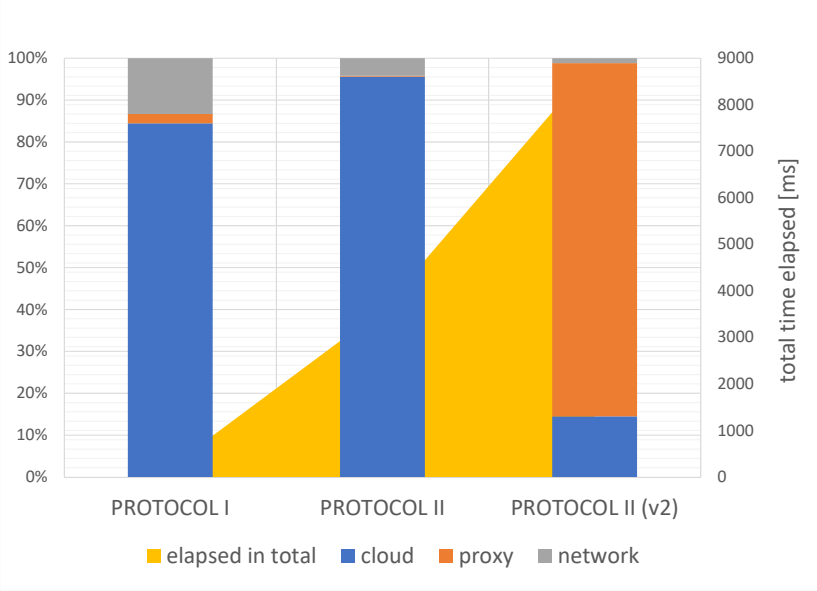

Fig. 4.2: Protocol stages brake down

Summary

As expected, additional complexity of protocol II (and inherently its variant) asks a heavy toll. The fact that each protocol execution requires multiple requests causes a non-negligible overhead. Also the servers can not compute their partial result on their own, they need to *wait* for the peer server to supply its fragment (after it's obfuscated). Despite the total number of multiplication operations is only *five* (in case of two fragments), compared to *three* for protocol I, the significant overhead combined with the additional randomness introduction and permutation causes protocol II not to perform as well as protocol I.

When comparing protocol II and its variant, we can observe a very significant difference, even though the execution algorithm is basically the same. The crucial difference can be seen in the brake-down chart. Transferring more than third of all the computations to the proxy itself is very disadvantageous. Yes this result is heavily influenced by used hardware (being an old mid-range PC desktop) as well as the server configuration, so the absolute value might not be much representative. However, even if powerful machine is used with properly configured server, the role of a proxy is not to handle such a significant part of the entire computation. In case thousands of requests are issued in one minute, a "mere third" of one calculation can quickly translate to entire seconds of delay, waiting for resources to be available.

4.2 Arithmetic Mean Computation

In addition to the primary goal of this thesis – the multiplication protocols implementation – also a simple arithmetic mean computation request was implemented.

Data outsourced using a data splitting technique is bound to be easy to work with, without the need to download it each and every time. This was proven to be true in case of specific calculations, in our case multiplication. Even though many more operations could potentially be expressed as a protocol with specific steps, it is extremely useful to be able to access individual fragments without any additional effort. When comparing additive and vertical splitting, as explored in this thesis, only vertical splitting offers this important benefit.

Additive splitting in its principle changes values of the outsourced data, so it can not be accessed directly as a simple read request to the fragment holding server. To better compare the actual drawback, a simple mechanism to compute arithmetic mean of a column of the original data set was implemented. In case of protocol I, additive splitting forces us to download all three fragments, compute the sum and then extract the desired column to compute the mean. Amount of the downloaded data is three fold the original set and all the computation is done by the proxy.

By contrast, protocols II and its variant upload the data in the same format as is the original data set. This enables the proxy to let the calculation done by the server holding the fragment in question via a single request.

Protocol I	Protocol II	Protocol II variant
6642,5	302,25	316,5

Tab. 4.2: Measured average elapsed time [ms] - arithmetic mean

Average measured duration of the arithmetic mean computation requests can be seen in table 4.2. As expected, without the need to reassemble the data set, protocols II and its variant perform significantly better.

4.3 Storage

Since storage requirement is rather a static characteristic, it is safe to compare it even without implementation details. Each protocol's amount of stored data is basically influenced solely by the splitting technique and whether or not an additional data (e.g. permutation) needs to be stored by the proxy server. Moreover, taking into account proposed implementation, amount of temporarily stored data (i.e. cache), will be shown, as in production environment this could potentially cause drastic peaks in the system's dynamic memory usage. In case of both protocols, storing certain partial results is more than appropriate, given that it is requested more than once and the compute time is more valued.

Storage

As stated above, amount of disk space required is influenced solely by the splitting technique. In our case additive vs. vertical. Additive splitting is furthermore dependent on the target amount of fragments. In our scenario, three cloud servers are used and thus supplied data set is split into three equal parts: $X = a\ + B + C$. That means amount of outsourced data is *three* times as big. In case of more storage nodes, and more fragments, the data will grow linearly.

Contrary to additive splitting, vertical splitting does not increase the total data sizes. It is merely an extraction of a number of columns from the original data set. No matter how many fragments are needed, total data size will always stay the same as the original data set. This is a major benefit as disk space is an expensive commodity. Even a minor difference would play an important role, all the more a difference of hundreds of percents.

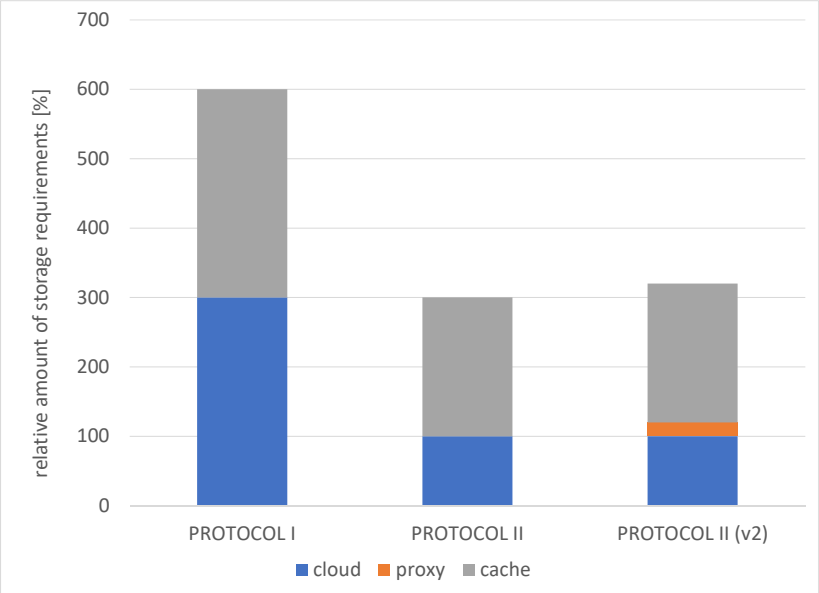

Fig. 4.3: Required disk space

Memory

In figure 4.3 we can see the relative disk usage comparison. 100% is the disk space required by the original data set. Apart from the storage requirements, effect on memory needs also to be discussed.

In case of protocol II (and its variant), caching the partial results is somewhat voluntary, but improves dramatically the performance. In case of protocol I however, to compute any of the partial multiplications, each cloud node requires two fragments. In case the two needed fragments are not found, the missing one needs to be transferred and temporarily stored (sufficiently enough in dynamic memory).

Splitting technique

Since splitting is applied to the data set only once, before uploading, importance of this process being done quickly is not that considerable. It is nonetheless an important part of the outsourcing protocol itself.

The comparison, measured during the same test scenario, can be seen on figure 4.4. As expected additive splitting is significantly more demanding. Vertical splitting is, after all, only a copy operation on the data stored in dynamic memory. Additive splitting however needs to process several (depending on the number of

outsourcing nodes) addition computations. Even though addition is not that demanding of an operation, given the size of data in question, this splitting technique can take several times more time than the simple vertical splitting.

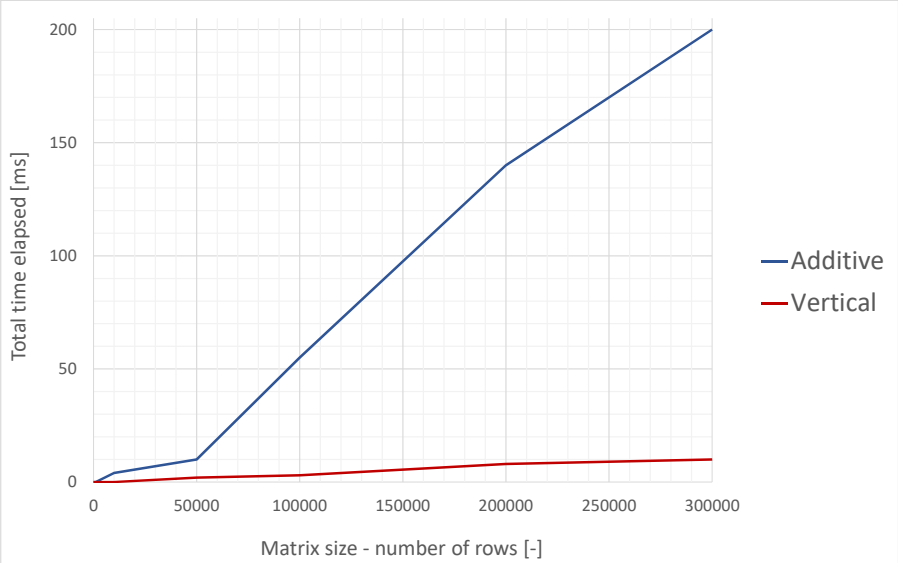

Fig. 4.4: Total time elapsed

Summary

As far as storage requirements go, the additive splitting technique causes a non negligible drawback. In case of three fragments, total amount of outsourced data will be three times more at best. In addition, missing fragments need to be transferred or stored beforehand in a redundant way, which even further increases final requirements on either memory or disk space up to *six times* the original amount.

5 Conclusion

In this thesis we dive into the problematic of cloud computing. Given the rising tendencies of the IT industry to use public clouds, it is without doubts a very interesting topic. The specific subject of this thesis is security. To be able to safely use public clouds as data storage, we must ensure, that the uploaded data is properly secured. A traditional way – encryption – is not suitable for public cloud scenarios because it is generally intended that various operations can be performed on the outsourced data, such as arithmetic computations (e.g. statistical analysis) or ranked searches. That is why in our thesis we further explored another, non cryptographic way – *data splitting*.

We base our efforts on an already published papers [11] and [15], dealing with secure outsourcing of matrix computations. We followed up on these publications with implementation of therein proposed protocols for matrix multiplication. We also took inspiration in the presented architecture – secure proxy, security as a service. In addition we proposed our original enhanced protocol. Enhanced in a sense of additional security. This protocol is to be secure even in *colluding* scenarios, i.e. cases where cloud service providers decide to cheat and cooperate to retrieve the original data.

Finally, this implementation was achieved with use of Java EE platform and deployed using Apache Tomcat and public cloud platforms to simulate real world conditions. Each protocol's performance was thoroughly measured using both real exemplary data set and randomly generated data. From the obtained results, we can observe a clear performance difference between the two protocols. The use of additive splitting offers more simple approach to the multiplication computation and thus the protocol I does achieve better results. However the inability to work with the uploaded data as is (in the thesis demonstrated on arithmetic mean computation) is a substantial shortcoming that convinced us to base our proposed variant on protocol II – vertical splitting. Further benefit of this approach are significant disk space savings.

Our proposed variant, based on protocol II, introduces additional permutation operation to provide more security. Due to this, in our implementation, more workload had to be transferred to the secure proxy, i.e. private infrastructure. During measurements, we observed that the execution was further slowed down, cause being the increased load on the proxy server. However, this variant, thanks to the secret permutation, proves to be effective in colluding scenario, which is a significant benefit. In real application, this crucial fact could potentially redeem its inferior performance.

Bibliography

[1] Internet World Stats [online]. Available at: <https://www.internetworldstats.com/>

[2] The World Bank [online]. Available at: <https://data.worldbank.org/indicator/it.net.user.zs>

[3] VNI Global Fixed and Mobile Internet Traffic Forecasts. Cisco [online]. Available at: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>

[4] CHENCHUN-YANG, Zhang, Chen C.L.PHILIP and Zhang CHUN-YANG. 2013. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Macau, China. University of Macau.

[5] FOOTE, Keith D. A Brief History of Cloud Computing [online]. Available at: <http://www.dataversity.net/brief-history-cloud-computing/>

[6] RightScale [online]. Available at: <https://www.rightscale.com/lp/2017-state-of-the-cloud-report>

[7] CIMINO, Steve. Cloud computing outages: What can we learn? [online]. Available at: <https://searchcloudcomputing.techtarget.com/feature/Cloud-computing-outages-What-can-we-learn>

[8] CRAWLEY, Kim. 6 Top Cloud Security Threats in 2018 [online]. Available at: <https://www.tripwire.com/state-of-security/security-data-protection/cloud/top-cloud-security-threats/>.

[9] MORROW, Timothy. 12 Risks, Threats, & Vulnerabilities in Moving to the Cloud [online]. https://insights.sei.cmu.edu/sei_blog/2018/03/12-risks-threats-vulnerabilities-in-moving-to-the-cloud.html

[10] RISTENPART, Thomas, Eran TROMER, Hovav SHACHAM and Stefan SAVAGE. 2009. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. San Diego, USA. University of California.

[11] NASSAR, Mohamad, Farida SABRI, Abdelkarim ERRADI and Qutaibah M. MALLUHI. 2013. Secure Outsourcing of Matrix Operations as a Service. IEEE CLOUD. 2013, 918-925.

[12] GANAPATHY, V., D. THOMAS, T. FEDER, H. GARCIA-MOLINA and R. MOTWANI. 2012. Distributing data for secure database services. Transactions on Data Privacy. Transactions on Data Privacy. 5(1), 253–272.

[13] ARMBRUST, Michael, Armando FOX, Rean GRIFFITH, et al. 2010. A View of Cloud Computing. Communications of the ACM. 2010(4), 50-58.

[14] Clarus: User centred privacy and security in the cloud [online]. Available at: <https://www.clarussecure.eu/>

[15] DOMINGO-FERRER, Josep, Sara RICCI and Carles DOMINGO-ENRICH. 2018. Outsourcing Scalar Products and Matrix Products on Privacy-Protected Unencrypted Data Stored in Untrusted Clouds. Information Sciences. 2018, 320-342.

[16] Apache Tomcat [online]. Available at: <https://tomcat.apache.org/

[17] Efficient Java Matrix Library [online]. Available at: <http://ejml.org/

List of symbols, physical constants and abbreviations

CSP Cloud Service Provider
DoS Denial of Service
VM Virtual Machine
IaaS Infrastructure as a Service
PaaS Platform as a Service
SaaS Software as a Service
SaaS Security as a Service
API Application Programming Interface
SLA Service level agreement
REST Representational State Transfer

List of appendices

A Computation Results

Computation results are presented here. Results were obtained using the mentioned data set (3.6) and all three presented protocols. Given the values of the column being:

Provider Id	Provider Zip Code	Total Discharges	Avg. Covered Charges	Avg. Total Payments	Avg. Medicare Payments
10005	35957	14	15131.85	5787.57	4976.71
...	...	...	...	...	...

arithmetic mean of columns *three* through *six* was computed as well as the $A^T \cdot A$ multiplication, i.e. correlation matrix of the data set.

A.1 Arithmetic Mean

Total Discharges	Covered Charges	Total Payments	Medicare Payments
42,7763	36133,9542	9707,4738	8494,491

A.2 Correlation Matrix

$$\begin{bmatrix}
14396578658528476 & 1887488881378786 & 1783256598089 & 1408378173935740 & 396965747469900 & 345902141822144 \\
1887488881378786 & 501249105333937 & 317716864695 & 304862618581816 & 76840832976152 & 67140790362564 \\
1783256598089 & 317716864695 & 724240040 & 245977881308 & 66689492688 & 57954419068 \\
1408378173936520 & 304862618582132 & 245977881540 & 413408094518455.56 & 91124169931692.69 & 82188234339521.8 \\
396965747468828 & 76840832976820 & 66689492980 & 91124169931692.69 & 24945917461917.965 & 22484735005444.59 \\
345902141822304 & 67140790362536 & 57954419716 & 82188234339521.8 & 22484735005444.59 & 20478413781078.523
\end{bmatrix}$$

B Execution Instructions

Here, instruction for proposed application execution are presented. As explained in the thesis, final application is divided into two parts: cloud server (computation endpoint) and local, secure proxy server. That is why, two distinct repositories were created, both are part of the appended CD: *cloud-api* and *broker-api*. Directories of the same name contain source code, written in Java 8, in form of a Maven project. Output of the build is a *.war* archive file, that was directly tested using Apache Tomcat 9.0.14 server. Both built packages can also be found on the appended CD, so there is no need to build the projects manually. Instructions for installation and execution can be further divided into two parts:

Broker API

This application is expected to be (and during presented measurements was) executed on a local machine. Standard way to deploy and execute the application, using the default Tomcat server configuration is:

1. Download the Apache Tomcat server from her: https://tomcat.apache.org/ or extract the appended archive, containing the exact version of the server used during our testing.
2. To be able to run Apache Tomcat server, Java Run-time Environment should be installed on the system with properly configured $JAVA_HOME variable (depending on used operating system). Additionally, $CATALINA_HOME variable needs also to be configured, pointing to the directory containing Apache Tomcat executable (https://tomcat.apache.org/tomcat-8.5-doc/appdev/installation.html).
3. Ensure that Apache Tomcat can be successfully run, by executing from the Tomcat root folder either *./bin/startup.sh* or *startup.bat* and navigating to the configured port in a web browser. Default Tomcat homepage should be loaded.
4. Once server functionality is tested, server can be shutdown and proposed broker-api application can be deployed. Easiest way is to copy the *broker-api.war* archive file into `$CATALINA_HOME/webapps` folder. Once start up in initialized, Tomcat searches for archives in this location and automatically explodes and deploys them.
5. Once deployed and started up, a rudimentary front end of the application can be accessed via web browser on the root URL of the application: `http://localhost:8080/broker-api`

For sake of additional convenience an simplicity, *.tar* archive file including tested version of Apache Tomcat server with the broker application already deployed. Provided that Java 8 JRE is installed and $CATALINA_HOME variable is set up (depending on th eoperating system), simple `./bin/startup.sh` from the root of the extracted folder will start up the server with the application running in the */broker-api* context.

Cloud API

As both parts of the service are built using the same technologies, the deployment of the cloud-api application is in its principle same as in case of broker API. Important difference is however the host machine.

In our thesis, we used real public cloud platform services. Using either AWS, Microsoft Azure or Google Cloud, we spawned 3 basic Linux server VMs and deployed our application there. Due to free-tier limitations of used accounts, deployed cloud applications used during our measurements are not guaranteed to be running for the potential reviewer of this thesis. In case they are still up and running, these test cloud-api applications can be accessed on following public URLs:

- 35.242.247.210:8080/test
- 34.90.82.161:8080/test
- 34.90.241.54:8080/test

In case they are not, potential reviewer is required to use at least three additional Linux servers and deploy the *cloud-api.war* application there, using the same process as described above. Note, that there is no need for these machines to be spawned in a public cloud, these can be locally started virtual machines or even the host machine (same as where broker-api application is running) itself. Correct target host URLs need however to be configured via application's homepage (root endpoint).

Again for sake of simplicity, *.tar* archive with Tomcat server and deployed cloud-api application is included in the appended CD. The server is configured in the default way, so if more than one is wanted to be deployed on the same machine, configuration needs to be modified not to cause port collision.

C Content of the appended CD

As described in appendix B, source code directories (versionned using *git*) as well as built packages and server binaries are included in the appended CD. Contents are as follows.

```
/ ..... root folder
├── source/ ..... git source code repositories
│   ├── broker-api/
│   │   └── ...
│   └── cloud-api/
│       └── ...
├── target/ ..... built .war packages and server binaries
│   ├── broker-api.war
│   ├── cloud-api.war
│   └── apache-tomcat-9.0.14.tar.gz
├── predefined/ ..... Tomcat server binaries with deployed applications
│   ├── broker-server.tar.gz
│   └── cloud-server.tar.gz
├── dataset.csv ..... example data set
├── dataset_clean.csv ..... clean, numerical only, example data set
└── thesis.pdf ..... text of the thesis
```